Web Services

Distributed Systems

Sistemi Distribuiti

Andrea Omicini *after Andrea Santi*
andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2013/2014

Outline

Disclaimer

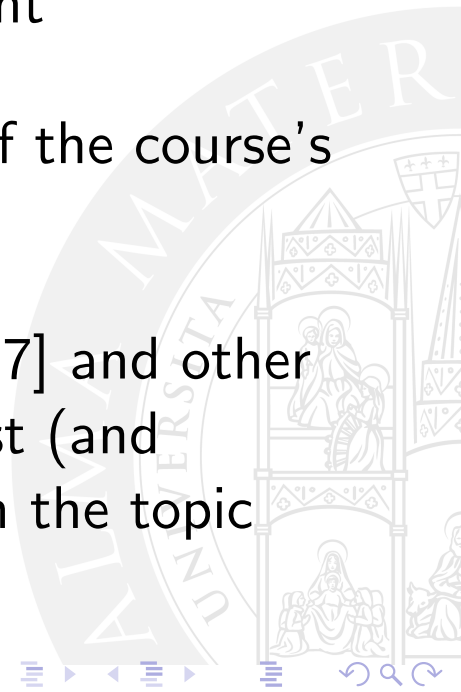
These slides were first developed by Andrea Santi

Every problem or mistake contained in these slides, however, should be attributed to the sole responsibility of the teacher of this course

Outline

Reference Material

- This presentation is rooted on some of the reference books on the topic [Erl, 2005, Richardson and Ruby, 2007]
- Most of the content of these slides has been re-adapted from the books [Erl, 2005, Richardson and Ruby, 2007] and integrated with new material according to a possibly different viewpoint
- Eventual mistakes/problems are the sole responsible of the course's professor
- The cited books [Erl, 2005, Richardson and Ruby, 2007] and other on-line documentation (e.g. [Oracle, 2011]) are a must (and recommended) read for a more comprehensive view on the topic

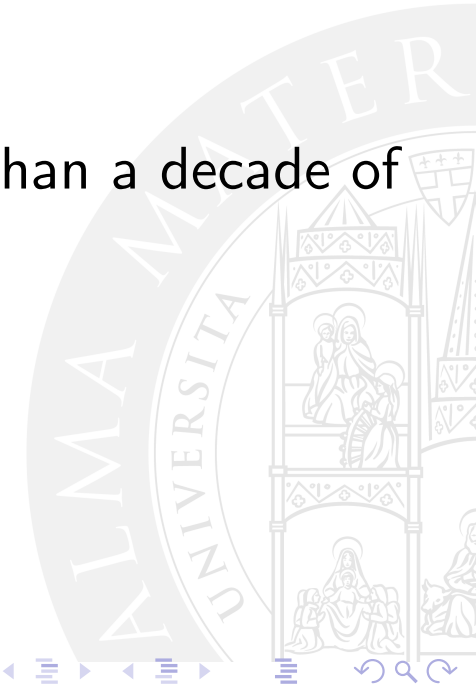


Outline

Outline

Web Services: One of the Buzzwords of the 21th Century

- Web Services caused some confusion in the IT world
- IT professionals, researchers, etc. claim their own interpretation
  - thus leading to troubles and misunderstandings
- There is not a *clear picture* on this topic after more than a decade of debate



So, Web Services: What are They?

- Good question, leading to a lot of other ones
- When to use them, and for what?
- Which architectural style should I use?
  - *Service-Oriented Architecture vs. Resource-Oriented Architecture*
- Is a web site a Web Service?
  - Even the answer to this question is now no more so clear [Richardson and Ruby, 2007]
- ...

Web Services: Tentative Definition I

Definition

Web Services (WSs) are client & server applications that communicate via *message-based interactions* over the World Wide Web's (WWW) HyperText Transfer Protocol (HTTP) [Oracle, 2011].

Web Services: Tentative Definition II

Main Features

- A WS encapsulates a unit of logic/functionality within a certain context
- The functionalities provided are described by a proper *contract*
  - explicit (SOA) vs. implicit (mostly, in ROA)
- Autonomy
- Loose coupling
- Composability
- Reusability
- Multi-vendor support and interoperability

Why Studying Web Services in this Course?

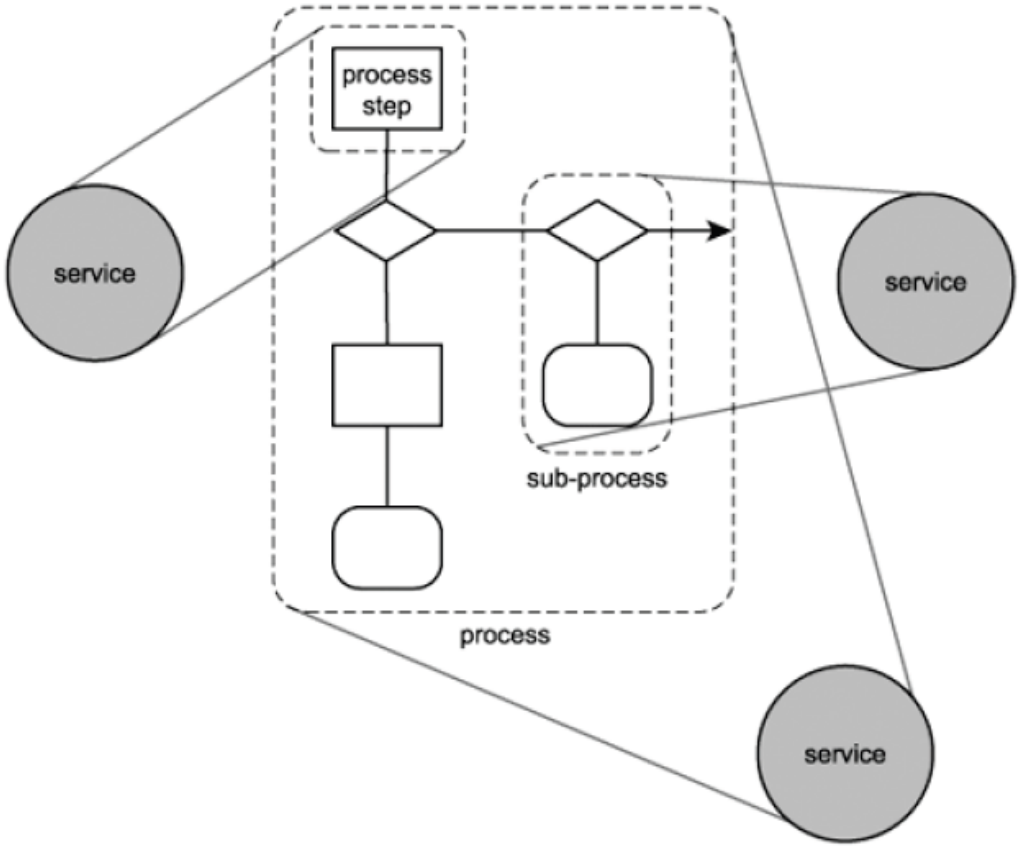
Nowadays Web Services are the reference stack of protocols for building *interoperable distributed systems*

- Enabling technology for different styles of communication
  - Message passing
  - Remote Procedure Call (RPC)
- Enabling interoperability thanks to a set of well defined standards
  - Between vendor-diverse applications
  - Between legacy and new applications

Outline

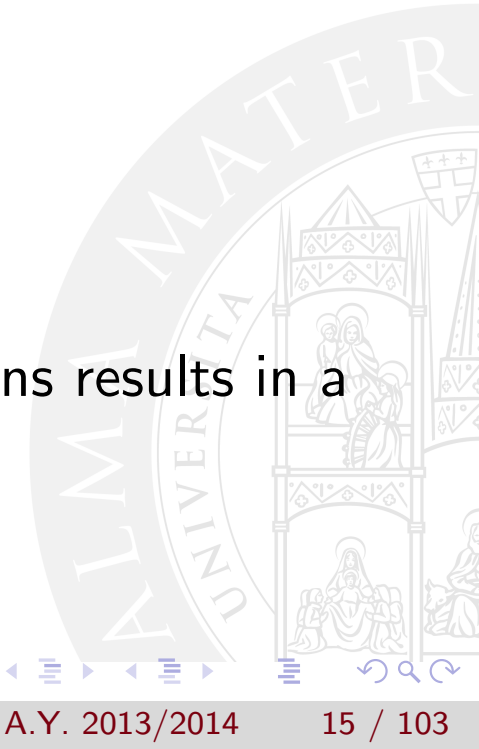
How do Web Services Encapsulate Logic?

- Web Services encapsulate logic within a distinct context
- This context can be specific to a business task, a business entity, or some other logical grouping



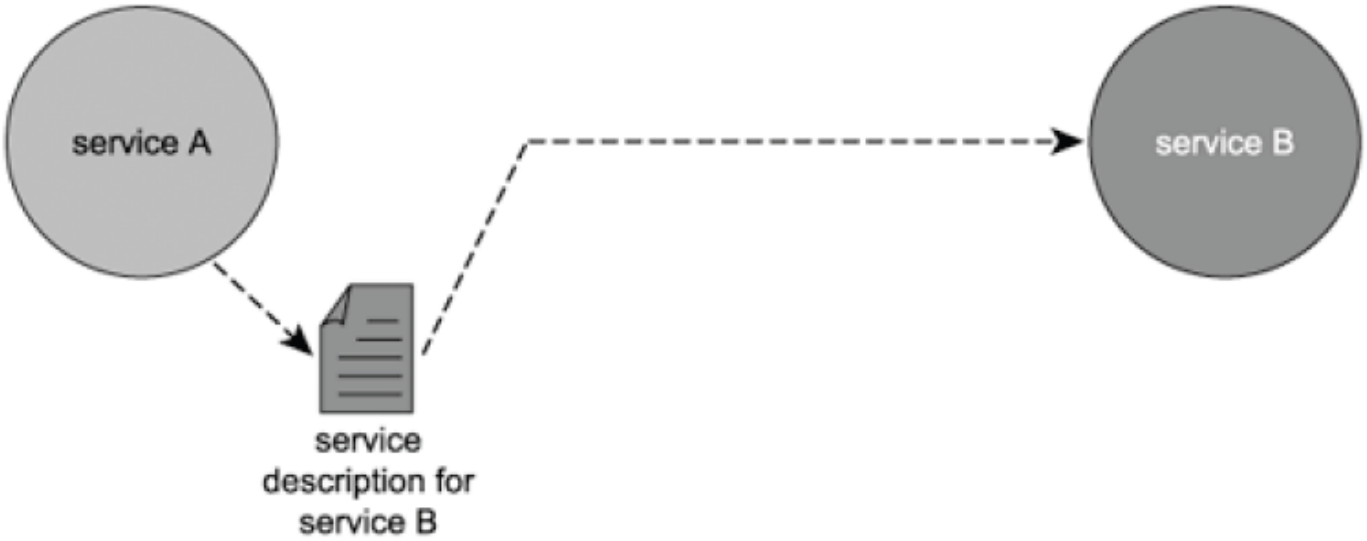
How do Web Services Relate to Each Other? I

- Web Services relationship is based on an understanding that for services to interact, *they must be aware of each other*
- This awareness is achieved through the use of *service descriptions*
- A Web Service description establishes (at least)
  - The *name* and the *address* of the Web Service
  - The *data expected* and *returned* by the Web Service
- The way in which Web Services use service descriptions results in a relationship classified as *loosely coupled*



How do Web Services Relate to Each Other? II

- Web Service *A* is aware of Web Service *B* because *A* knows *B*'s service description
- Knowing *B*'s service description, *A* has all of the information it needs to communicate with *B*

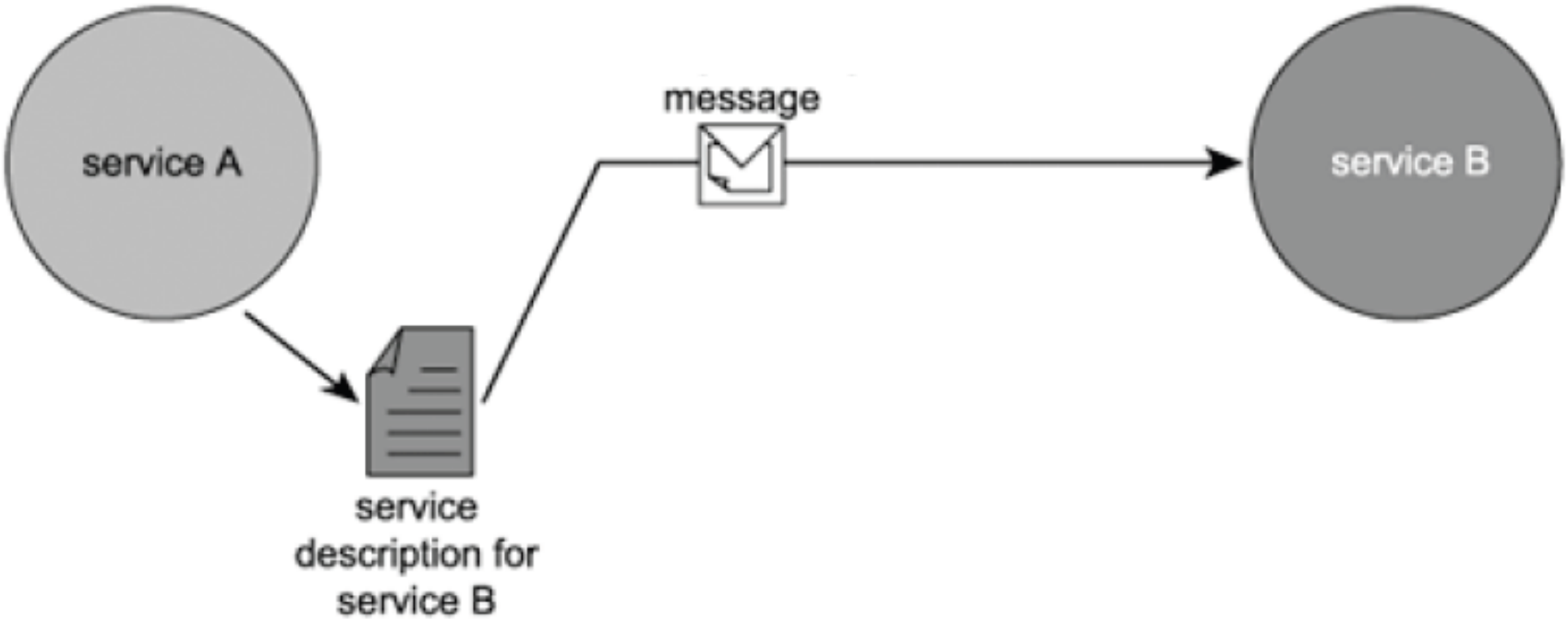


How Web Services Communicate? I

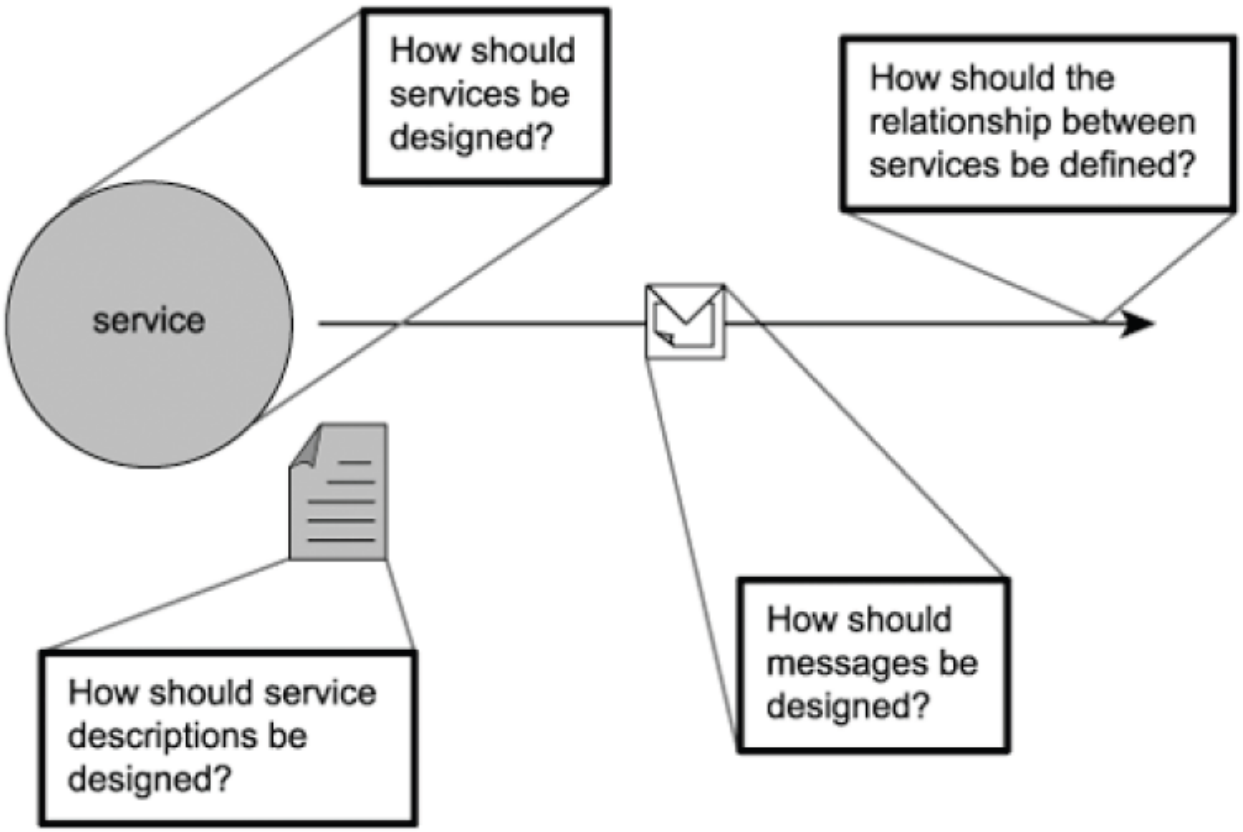
- Web Services communicate by means of proper *exchanges of messages*
- After a Web Service sends a message on its way, it loses control of what happens to the message thereafter
  - Messages are *independent units of communication* [Erl, 2005]
- Supported styles of communication
  - Asynchronous communication
  - Synchronous communication

How Web Services Communicate? II

A simple communication example

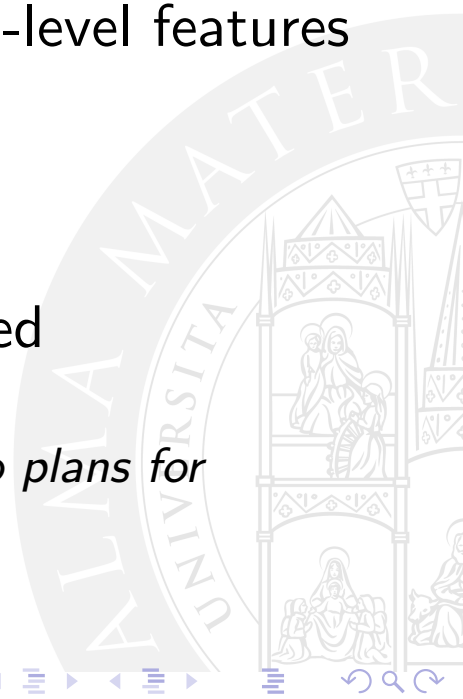


How to Design Web Services?



Two Different Architectural Approaches

- Service Oriented Architecture (SOA)
  - HTTP — as the *underlying* transport protocol
  - SOAP — as the *real* transport protocol
  - WSDL — for service description
  - XML — for formatting the messages exchanged
  - WS-* — set of specifications for handling high-level features
- Resource Oriented Architecture (ROA)
  - HTTP — as the real transport protocol
  - XML — for formatting the messages exchanged
  - WADL — for service description
    (standard submitted by Sun/Oracle to W3C, no plans for approving it)



Outline

Outline

What is a Service Oriented Architecture (SOA)?

A *"formal"* definition

SOA can be defined as an *open, agile, extensible, federated, composable* architecture *comprised of autonomous, QoS-capable, vendor diverse, inter-operable, discoverable*, and *potentially reusable* services [Erl, 2005]

Main features of the Service-Oriented Architectural model

- A service encapsulates a unit of logic within a certain context
- Loose coupling and message-based interactions
- Autonomy
- Composability
- Reusability
- Multi-vendor support and interoperability

Well, wait. . . something sounds familiar. . .

- Do you find any similarities with the Web Service definition provided a few slides ago?
- We are using two terms for referring to the same thing?
  - No. . .
- So, Web Services $\leftrightarrow$ SOA-based application?
  - Partially true but. . .

SOA & Web Services: Let's Make Things Clear I

- SOA and Web Services are not synonyms!!!
  - The former it's a *definition* of an architecture (principles, features...)
  - The latter is a *concrete implementation* of the service-oriented architectural model
- Web Services are the *reference* framework providing a *concrete implementation* of the service-oriented architecture
- A WS-based application is *not necessarily* a SOA-based application
  - WS realized exploiting the ROA approach
  - WS used just for enabling RPC
    - A SOA-based application must adhere to the basic SOA features (e.g. loose coupling, service autonomy, etc.)

SOA & Web Services: Let's Make Things Clear II

- Why all this confusion then?
- *SOA is intrinsically reliant on Web services so much so that Web services concepts and technology used to actualize service-orientation have influenced a number of the SOA characteristics identified before* [Erl, 2005].
- But in reality the features we have described in *"Web Service Fundamentals"* are SOA founding features
  - Supported by the reference implementation of the service-oriented architectural model

Outline

Designing SOA-based Web Services

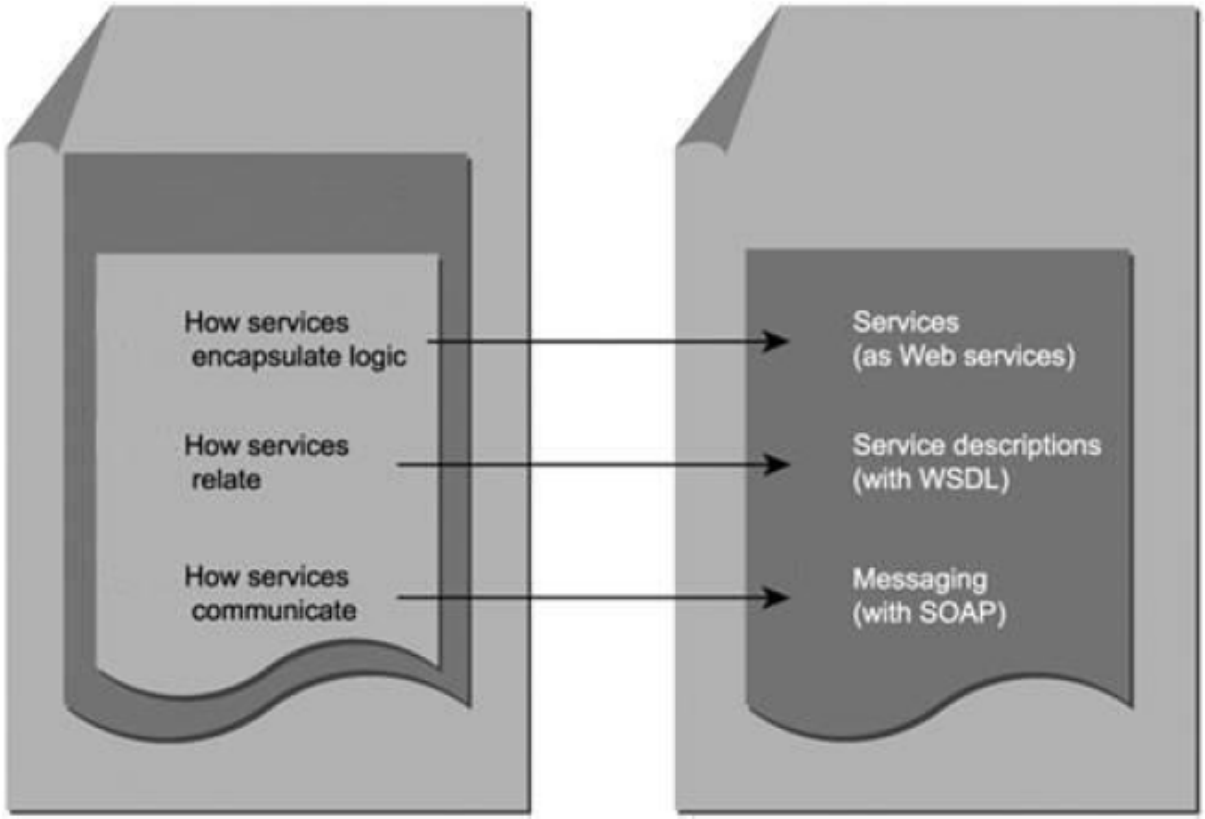


Figure : Mapping of SOA concepts into the WS framework [Erl, 2005]

A Guiding Example: An "Hello World" Web Service

- Classical entry-level example
- One Web Service that prints in standard output the message "Hello X" where X is the person/thing to greet

Services (as Web Services)

Web Service (WS): main features

Technological abstraction used for concretely implement a *Web Service* in a service-oriented fashion

A Web Service can be associated with...

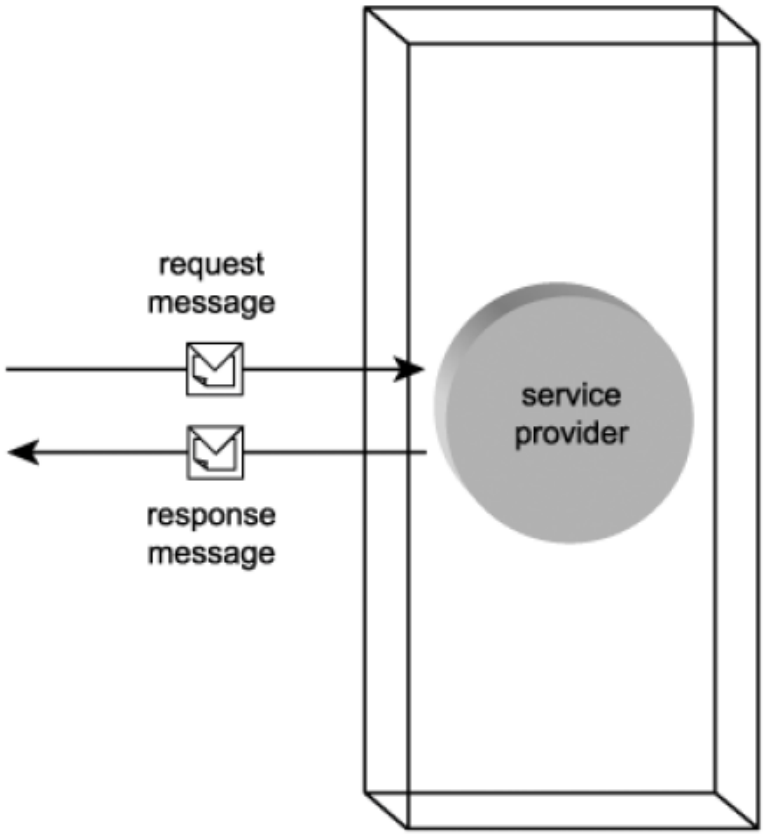
A service role — runtime classification depending on its responsibility in a given scenario (initiator - requestor - intermediary)

A service model — permanent classification depending the role played by the WS into an application (broker - utility service...)

Service Provider Role

A WS recipient of a request message is classified as a *service provider*

- The WS is invoked by an external source
- The WS provides a published service description (WSDL)



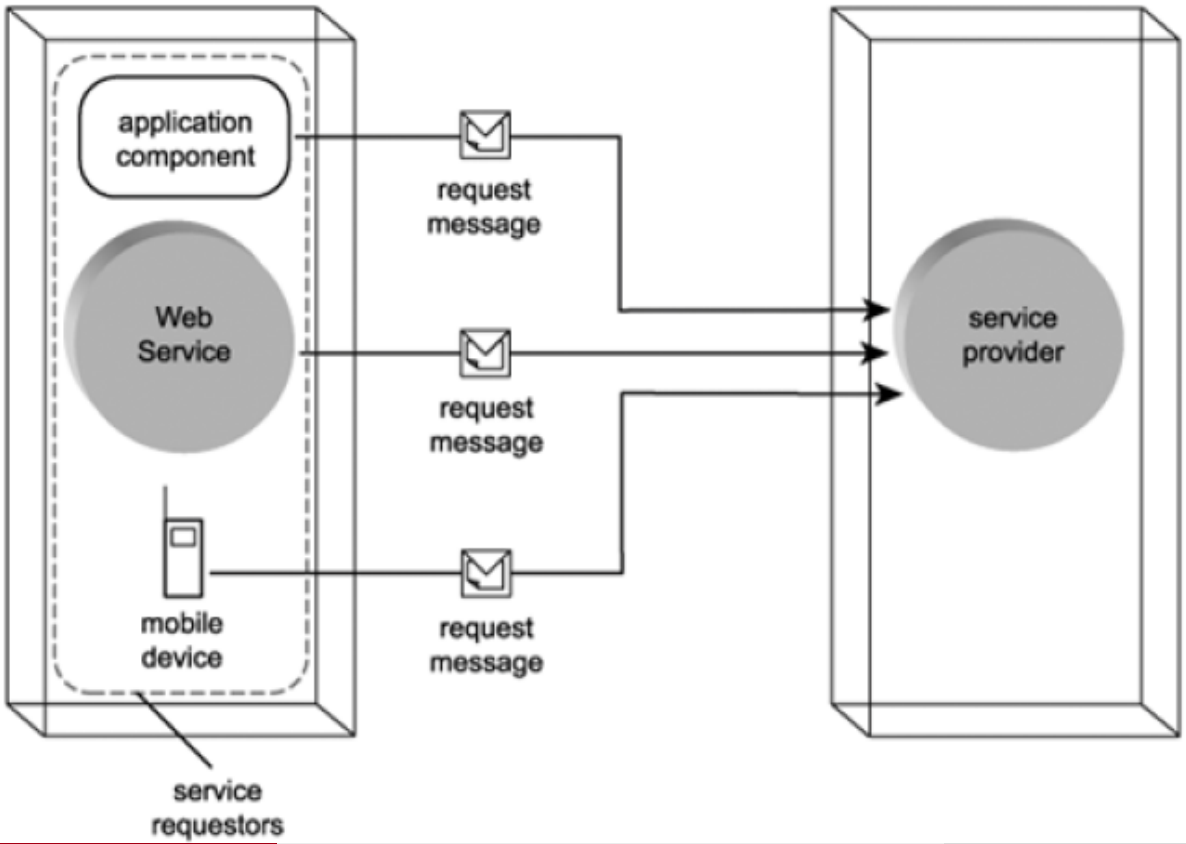
Service Provider in Our Example

- The `HelloService` Web Service is the service provider
  - It provides the basic greeting service
- Requests an input message containing the person/thing to greet
- Provides as output a message containing the greeting

Service Requestor Role

The sender of a request message is classified as a *service requestor*

- The requestor searches for the most suitable service provider studying available service descriptions
- The requestor invokes a service provider by sending to it a message



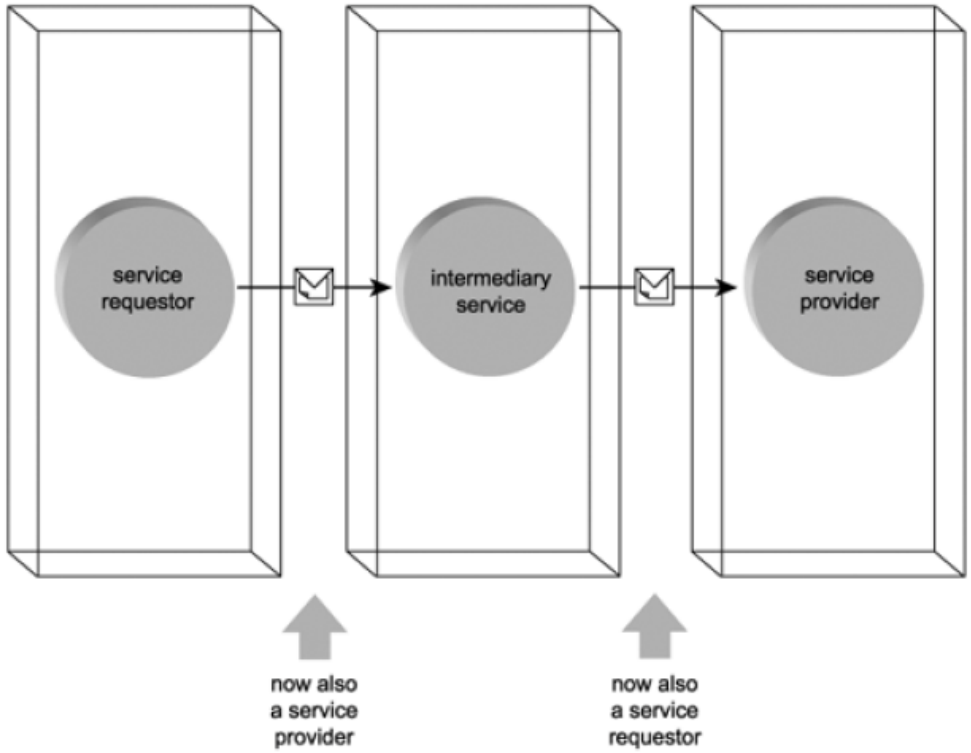
Service Requestor in our Example

- The Java application exploiting the `HelloService` Web Service
- Invokes the Web Service providing the appropriate input message
- Retrieves the desired response message

Service Intermediary Role

A message can be processed by multiple intermediaries before its final destination

- Passive intermediaries: simply route messages
- Active intermediaries: route messages to a forwarding destination actively processing/altering the message contents



Simple Object Access Protocol (SOAP) I

What is it?

The standard transport protocol for messages exchanged by Web services

- HTTP is used as the *underlying* transport protocol for SOAP messages

- Originally designed to replace proprietary RPC protocols (i.e. serialization of object)
- Now, despite the name, is a way to define a standard message format
  - Important remark: others transport protocols can be used as well
- Extremely flexible and extensible
  - Has been revised several times to accommodate more sophisticated features and message structures

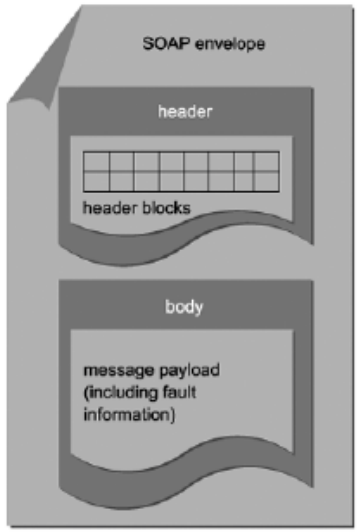
Simple Object Access Protocol (SOAP) II

Structure of a SOAP message

envelope — the message container: houses all the message parts

header — dedicated to hosting meta-information (used by WS-* specifications, described next)

body — the message content (i.e. XML-formatted data)



The SOAP Request Message in Our Example

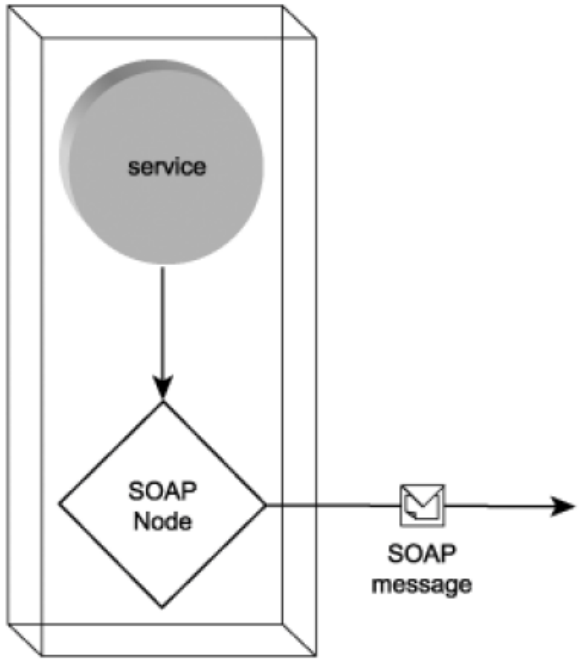
```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:sayHello
           xmlns:ns2="http://helloservice/">
            <arg0>John</arg0>
        </ns2:sayHello>
    </S:Body>
</S:Envelope>
```

The SOAP Response Message in Our Example

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
        <ns2:sayHelloResponse
          xmlns:ns2="http://helloservice/">
            <return>Hello, John.</return>
        </ns2:sayHelloResponse>
     </S:Body>
</S:Envelope>
```

SOAP Nodes

- WS are self-contained units of processing logic, but they are reliant upon a physical communication infrastructure
- Every platform has its own implementation of SOAP communications
- In abstract, the programs that services use to transmit/receive SOAP messages are referred as *SOAP nodes*



Web Service Description Language (WSDL) I

- XML-based language used for defining *service descriptions*
- A WSDL document define
  - The functionalities provided by the service
  - The service behavior

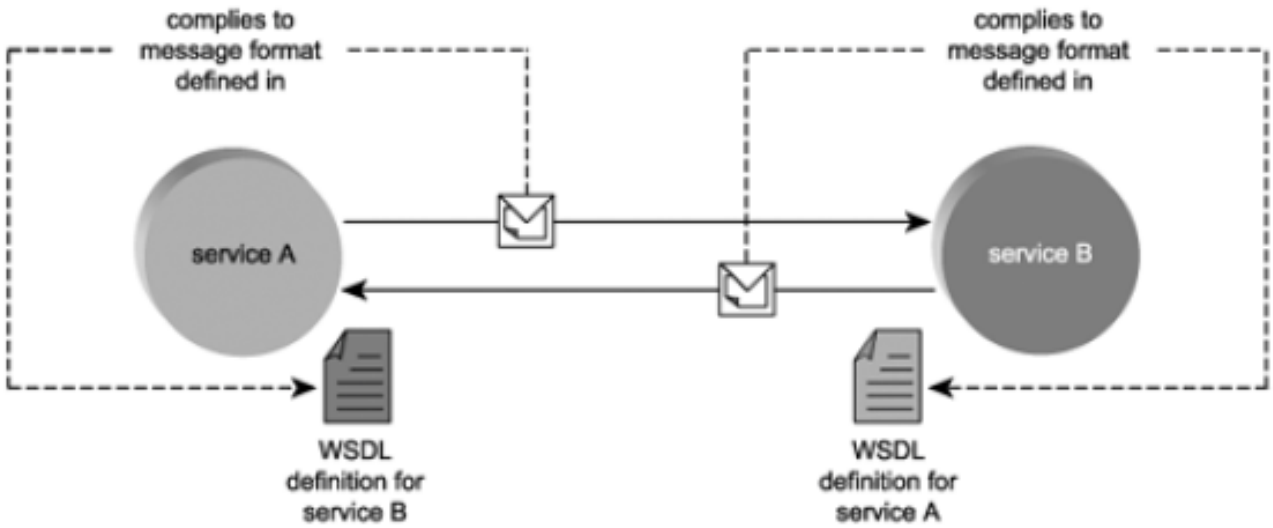


Figure : WSDL definitions enable loose coupling between services

Web Service Description Language (WSDL) II

Parts of a WSDL document

A WSDL service description is composed of two parts

- An abstract description
- A concrete description

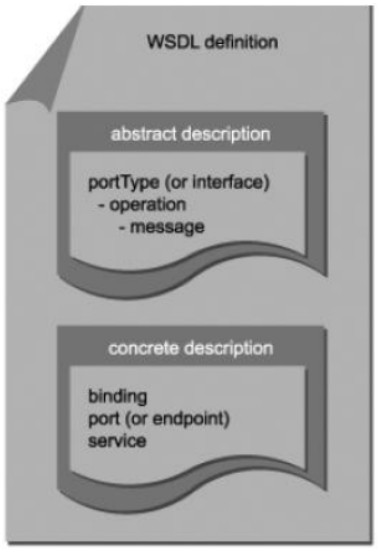


Figure : A WSDL document abstract representation

WSDL Abstract Description

Abstract description purpose

Establishes the interface characteristics of the Web Service without any reference to

- The technology used to implement the Web Service
- The technology used to transmit/receive messages

Abstract description elements

portType is a high-level view of the service interface by sorting the *messages* a service can process into groups of functions known as *operations*

operation is a specific action performed by the service

message is the abstraction used for describe operation's input/output

WSDL Concrete Description

Concrete description purpose

Establishes the physical connection (*binding*) of the WSDL abstract description to a physical transport protocol

Concrete description elements

binding describes the requirements (i.e. the transport protocol) for establishing a physical connection with the Web Service

service define the WS name and the set of service *port*s (i.e. all the possible service contact addresses)

port is the physical address at which a service can be accessed with a specific protocol

SOAP & WSDL

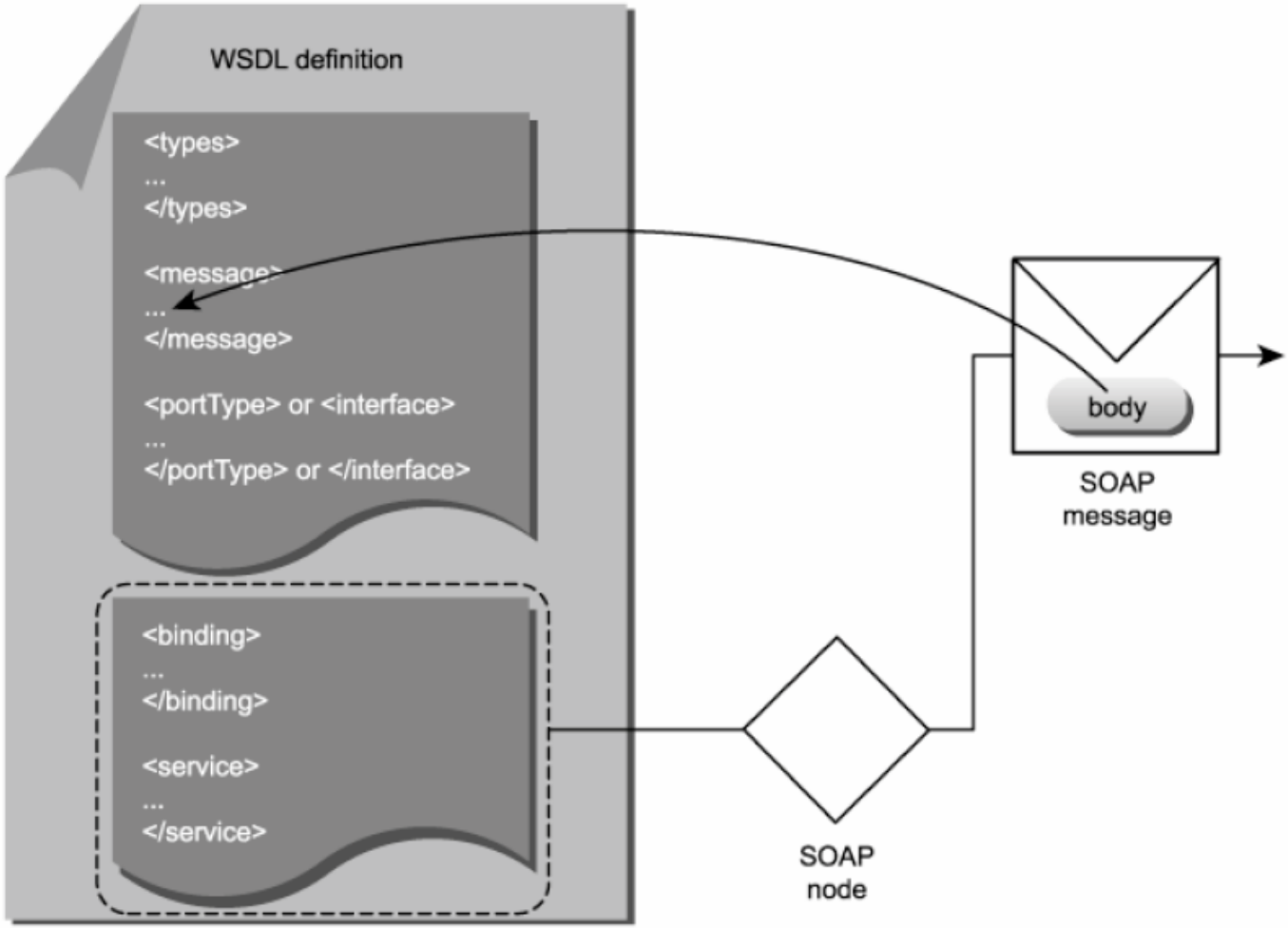


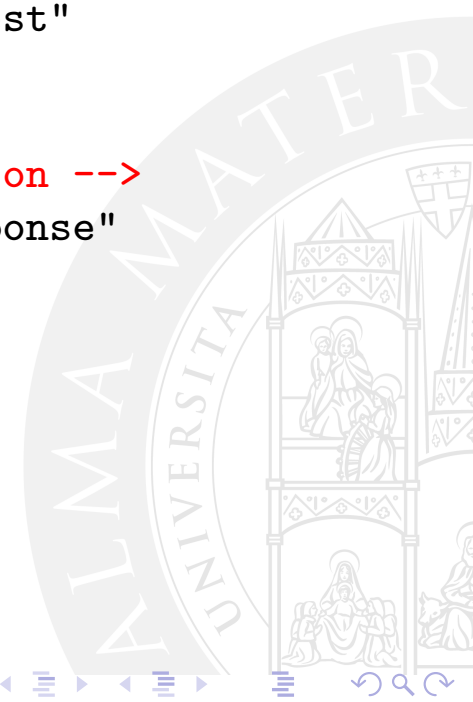
Figure : Relation between a SOAP message and its related WSDL document

The WSDL of the Hello Service I

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- Generated by JAX-WS ..>
<definitions xmlns:wsp1_2="http://schemas.xmlsoap.org/ws..."
  xmlns:tns="http://helloservice/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://helloservice/" name="HelloService">
<!-- Import of the XML data-types used -->
<types>
  <xsd:schema>
    <xsd:import namespace="http://helloservice/"
      schemaLocation="http://localhost:8080/helloservice/HelloService?xsd=1" />
  </xsd:schema>
</types>
<!-- Messages definition-->
<message name="sayHello">
  <part name="parameters" element="tns:sayHello" />
</message>
<message name="sayHelloResponse">
  <part name="parameters" element="tns:sayHelloResponse" />
</message>
```

The WSDL of the Hello Service II

```
<!-- PortType definition-->

<portType name="HelloService">

  <operation name="sayHello">

    <!-- Definition of the input message for the Hello operation -->
    <input wsam:Action="http://helloservice/Hello/sayHelloRequest"
        message="tns:sayHello" />

    <!-- Definition of the output message for the Hello operation -->
    <output wsam:Action="http://helloservice/Hello/sayHelloResponse"
        message="tns:sayHelloResponse" />

  </operation>

</portType>
```



The WSDL of the Hello Service III

```
<!-- PortType binding definition -->
<binding name="HelloServicePortBinding" type="tns:Hello">
  <soap:binding transport="http://..." style="document" />
  <operation name="sayHello">
    <soap:operation soapAction="" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
  </operation>
</binding>
```

The WSDL of the Hello Service IV

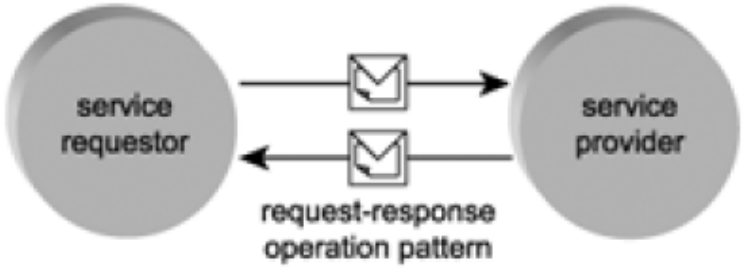
```
<!-- Service definition -->
<service name="HelloService">
  <port name="HelloServicePort" binding="tns:HelloServicePortBinding">
    <!-- Service address -->
    <soap:address location="http://localhost:8080/helloservice/HelloService" />
  </port>
</service>
</definitions>
```

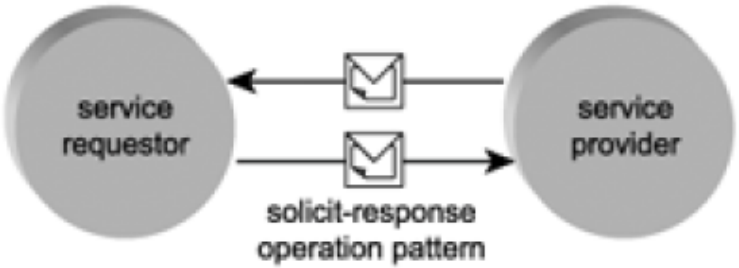
Message Exchange Pattern (MEPs)

- Definition of all the possible interaction dynamics between Web Services
- Group of already mapped out sequences for the exchange of messages
- Similar to design patters in software engineering, but oriented to *message exchange dynamics*
- WSDL 1.1 Supported MEPs
  - Request-Response & Solicit-Response
  - One-way & Notification
- WSDL 2.0 Supported MEPs
  - Old MEPs, but with new names and..
  - Basic MEPs + optional in/out or fault message

WSDL 1.1 Supported MEPs I

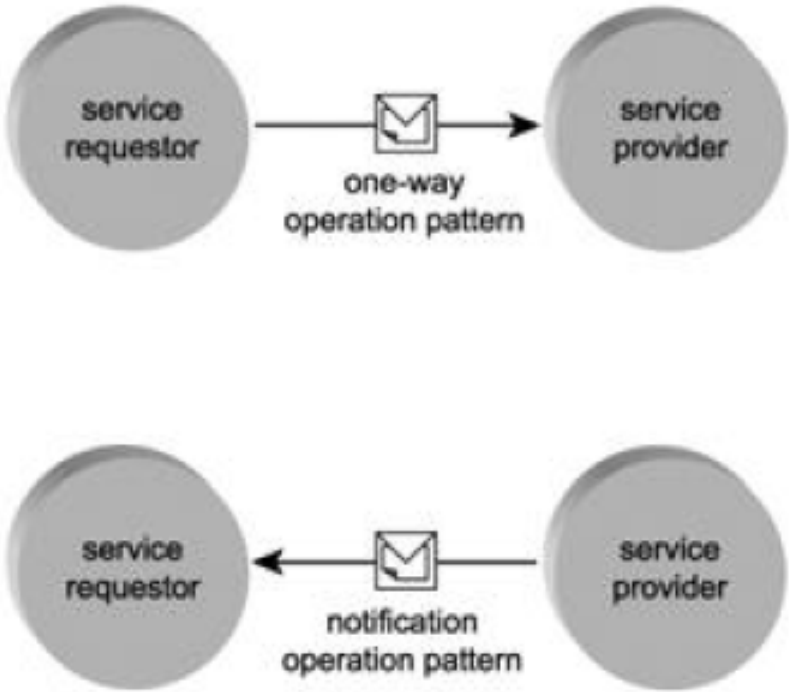
- Request-Response
- Solicit-Response





WSDL 1.1 Supported MEPs II

- One-way
- Notification



WSDL 2.0 Supported MEPs

Old MEPs, but with new names

In-out	equivalent to the Request-Response pattern
Out-in	equivalent to the Solicit-Response pattern
In-only	equivalent to the One-way pattern
Out-only	equivalent to the Notification pattern

New MEPs, introduced by WSDL 2.0

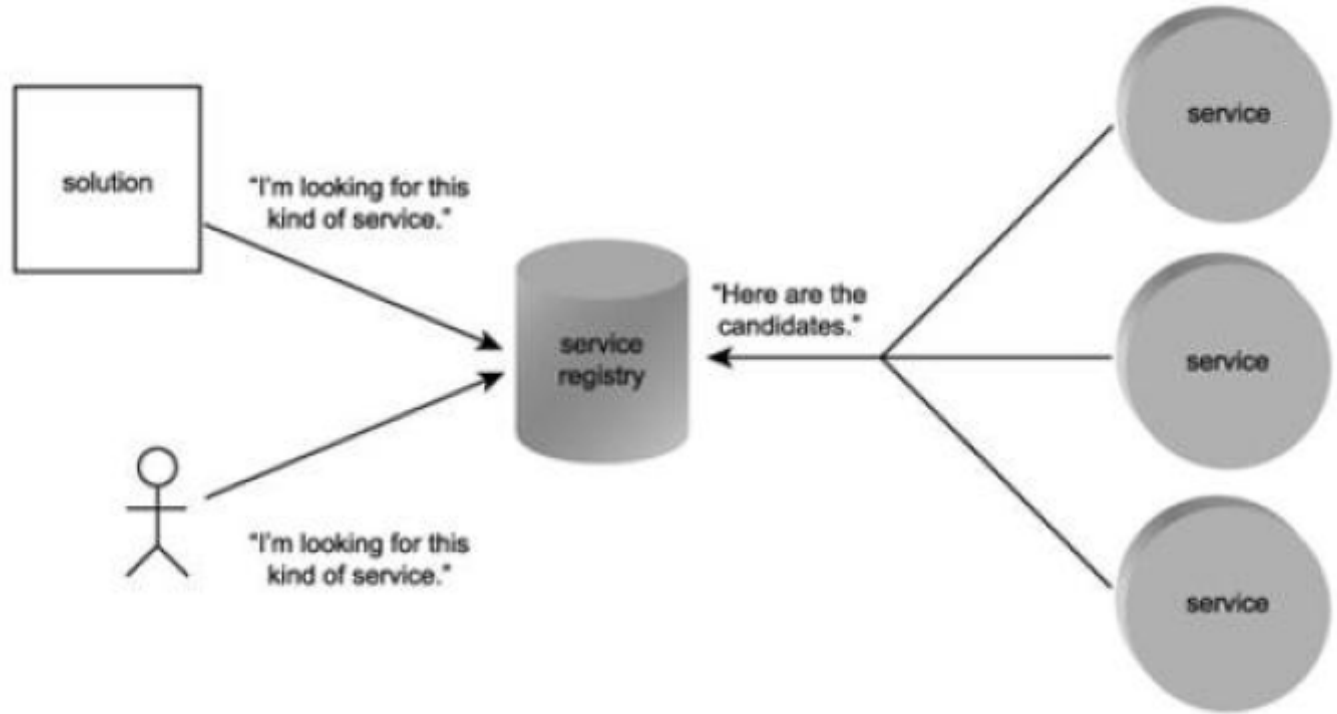
Variations of the basic four MEPs, in addition provides optional in/out message or fault response message

- Robust in-only
- Robust out-only
- In-optional-out
- Out-optional-in

Universal Description Discovery and Integration (UDDI)

OASIS standard that *tries* to address the issues related to service discovery and composition

- Functionalities advertising by registering the WSs' WSDLs into the UDDI registry
- Service requestors search functionalities offered by Web Services simply querying the registry



UDDI Problems and Limitations

Main problems: no semantics aspects are considered

- Without addressing semantic issues Web Service discovery and composition can not be successfully handled
- UDDI service advertising/discovery only rely upon *syntactic aspects*
  - Full signature-match for an operation is required
  - Otherwise how could one infer that a functionality (i.e. a WS operation) such as *rent a vehicle* is related to functionality *rent a car*?

UDDI is not so much widespread yet

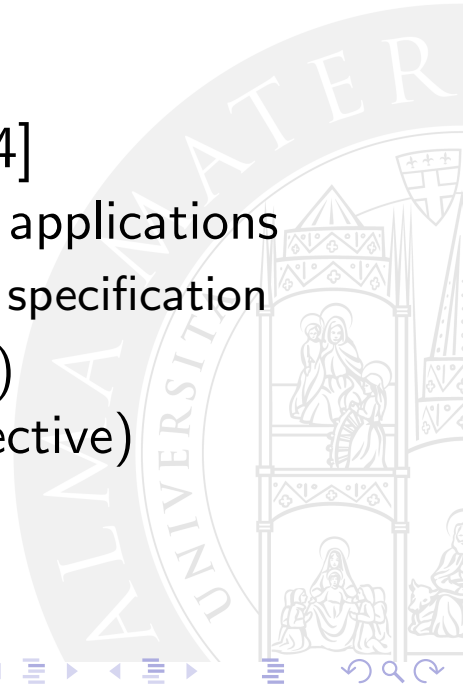
For taking advance of WS discovery and composition by means of UDDI are required

- A widespread diffusion of the public UDDI registries
- The registration of a high number of WSs

Outline

Web Service Tools Overview

- Java Metro (GlassFish) [Sun/Oracle, 2004]
  - Proposed as a *one-stop shop for all your web service needs*
    - From the simplest hello world web service...
    - ... to reliable, secured, and transacted web services that involves .NET services
  - Part of the GlassFish Application Server
- Apache Axis2 [The Apache Software Foundation, 2004]
  - Java platform for creating and deploying web services applications
    - Born from the Apache implementation of the SOAP specification
  - First version: Axis (RPC-perspective on Web Services)
  - New version: Axis 2 (Web Services in the SOA perspective)



JAX-WS (2.0): API Standardizations

- It is a specification...
  - so different implementations (e.g. Axis2, Java Metro,..)
- ... of a programming model (= set of API)
  - Java-based
- Aiming at simplifying the development of SOA applications through the support of a standard, annotation-based model to develop WSs and clients in Java
- Document-centric messaging model, replacing the remote procedure call programming model as defined by previous APIs
  - SOA perspective

Quick Overview of JAX-WS 2.0

- Simpler way to develop/deploy Web services
  - w.r.t. previous approaches, e.g. JAX-RPC
- Plain Old Java Object (POJO) can be easily exposed as a Web service
- Part of Java SE and Java EE platforms from Java 1.5
- Protocol and transport independence

Server-side: Two Basic Ways for Building Web Services

- Starting from a WSDL file (top-down approach)
  1. Generate required classes/sources using proper tools (e.g. **wsimport**)
     - WS interface
     - WS implementation skeleton class
  2. Add business logic to the generated WS implementation sources
  3. Build, deploy, and test the WS
- Starting from a POJO (bottom-up approach)
  1. Properly annotate the POJO
  2. Build, deploy, and test the WS
  3. WSDL file generated automatically starting from the annotated class

Server-side: an Example Starting From a WSDL

- Consider the `HelloService` WSDL of our sample
- Generate the sources starting from the WSDL
  - `wsimport -s <src path for gen sources> <wsdl path/URL>`
- Implement the WS business logic starting from the generated sources

```
public interface HelloService {
    @WebMethod
    @WebResult(targetNamespace = "...")
    @RequestWrapper(localName = "sayhello", targetNamespace = "...")
    @ResponseWrapper(localName = "sayhelloResponse", targetNamespace = "", className = "helloservice.SayhelloResponse")
    @Action(input = "http://.../sayhelloRequest", output = "http://.../sayhelloResponse")
    public String sayhello(
        @WebParam(name = "name", targetNamespace = "...")
        String name);
}
```

- Other command options
  - `-d`: Path for generated compiled classes
  - `-b`: Path to additional xml files defining WS used types
  - ...

Server-side: an Example Starting from a POJO

```
@WebService(serviceName = "CalculatorService")
public class CalculatorService {

   @WebMethod(operationName = "add")
   public java.lang.Double add(
     @WebParam(name = "firstParam") Double firstParam,
     @WebParam(name = "secondParam") Double secondParam) {

        return firstParam + secondParam;

   }
}
```

- **@WebService** annotation
  - Label the class as a Web Service
- **@WebMethod** annotation
  - Label methods as Web Service operation
- WSDL/Schema generated automatically

Client-side Programming I

1. The process for creating a Web Service client application always starts with an existing WSDL document
2. Point a tool (e.g. `wsimport`) at the WSDL for the service
   - `wsimport -s <src path for gen sources > <wsdl path/URL>`
3. The tool generates the corresponding Java source code for the described interface
   - JAXB used for providing WSDL $\leftrightarrow$ Java data-binding
4. Instantiate the generated WS skeleton class
5. Get a proxy using a `get<ServiceName>Port` method
6. Invoke any remote operations

Client-side Programming II

```
CalculatorService svc = new CalculatorService();
Calculator proxy = svc.getCalculatorPort();
int answer = proxy.add(35, 7);
```

- No need to use factories
- The code is fully portable
- XML is completely hidden from programmer

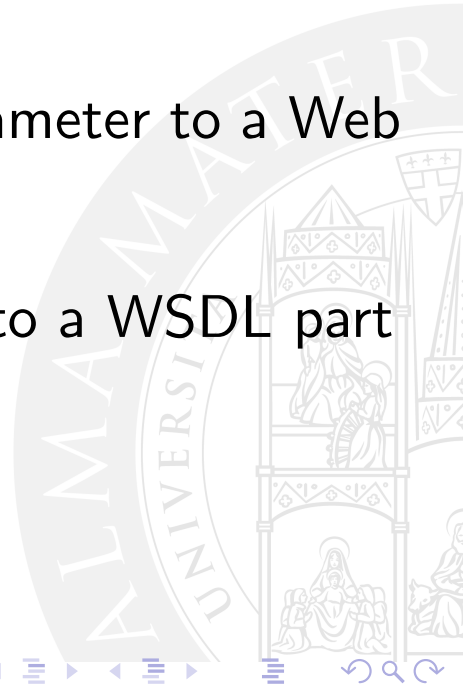
Principal Annotations

@WebService Marks a Java class as implementing a Web Service, or a Java interface as defining a Web Service interface

@WebMethod Customises a method that is exposed as a Web Service operation

@WebParam Customises the mapping of an individual parameter to a Web Service message part and XML element

@WebResult Customises the mapping of the return value to a WSDL part and XML element



Outline

The SOA/WS Evolution

The first WS generation introduced the framework building blocks and the basic specifications: WSDL, SOAP, UDDI...

The second generation of Web Service

With the second generation of WS has been introduced a set of specification (WS-*) for the managing of advanced functionalities:

WS-Coordination provides the rules for coordinating complex activities (AtomicTransactions, BusinessActivities) between WSs

WS-Security framework is a set of security specifications that provides authentication, authorization, data integrity and so on...

WS-BPEL defines a language for specifying business process behavior based on Web Services

... and many others WS-MetadataExchange, WS-Choreography, WS-Federation..

WS-Coordination

Main features

- Defines a general-purpose framework for managing complex activities
- Rooted on a general model for coordinating the common part of different complex activities
  - i.e different coordination activities can be coordinated using the same coordination model
- Aspects related to a particular coordination type are defined into a separated specification

Supported coordination types

Currently only two coordination types are supported

- WS-AtomicTransaction
- WS-BusinessActivities

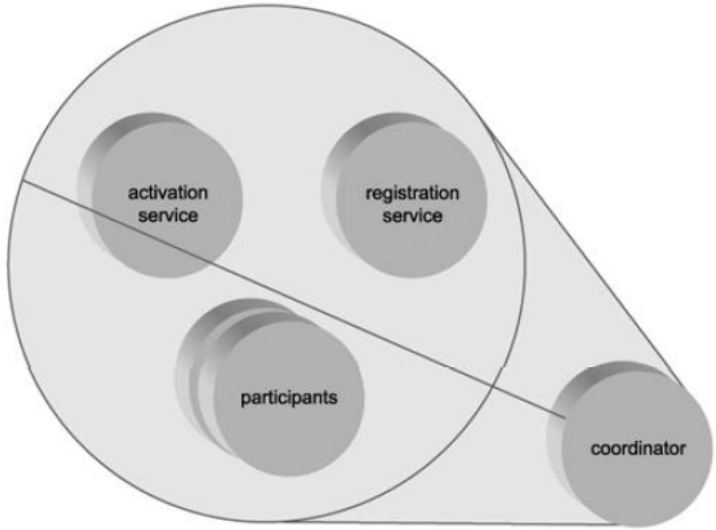
WS-Coordination General Model

Service involved

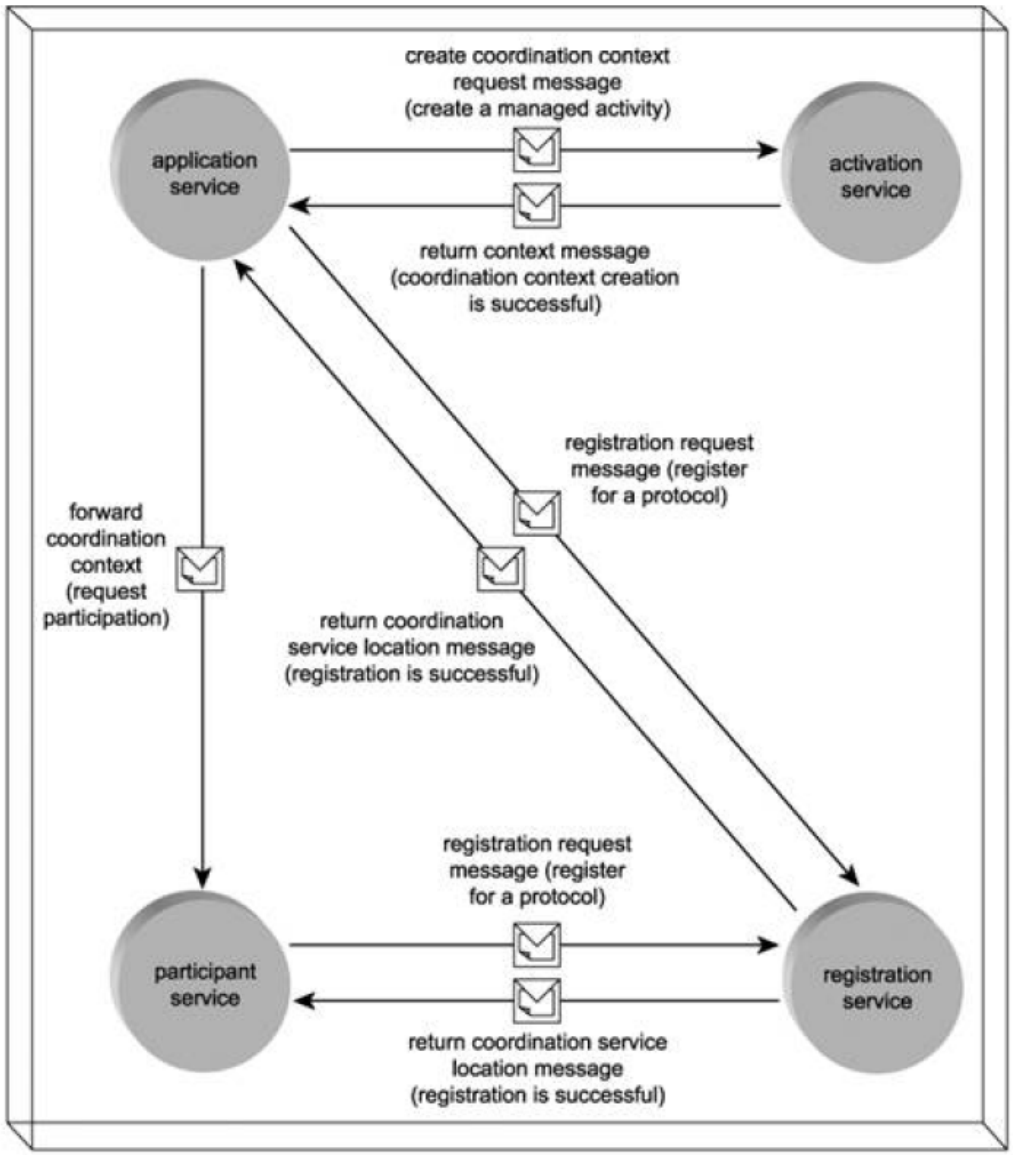
Activation Service responsible of the coordination-context creation (i.e. the identifier of the coordination activity)

Registration Service registers and keeps track of the participants of a complex activity

Coordinator Service manages the coordination of an activity w.r.t. a particular coordination type



WS-Coordination Dynamics Example

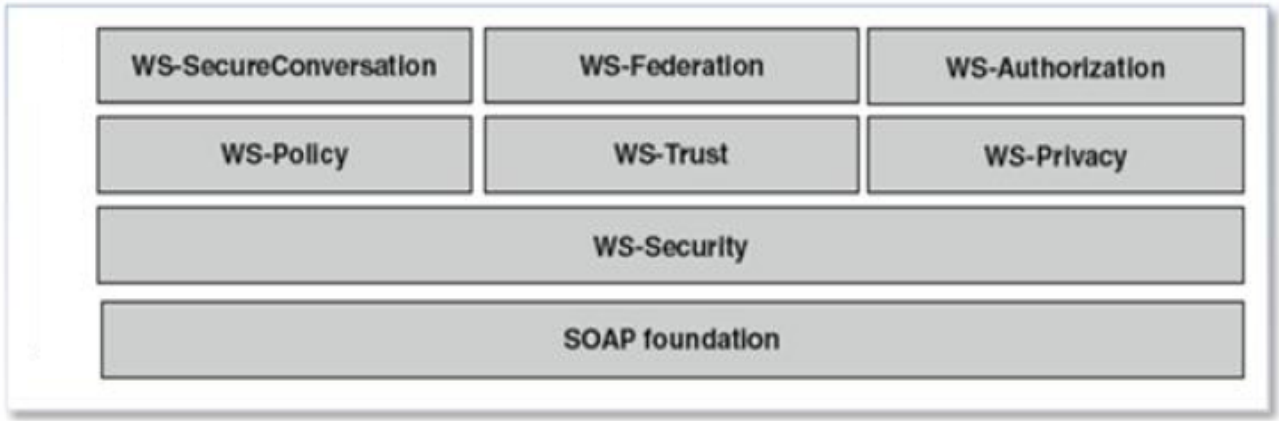


WS-Security Framework

A set of WS specifications that address almost all the issues related to Web Service security

Specifications belonging to the security framework

- WS-Security
- WS-Policy
- WS-Trust
- WS-SecureConversation
- Others...



WS-Security & WS-Policy

WS-Security

Enables applications to conduce secure SOAP message exchanges ensuring

- Message integrity
- Message confidentiality
- Message authenticity

Relies upon a set of existing specification:XML-Encription, XML-Signature..

WS-Policy

- Defines a general purpose model and corresponding syntax to describe the policies of a Web Service...
  - ...also security policies can be defined
- A policy can describe service requirements, capabilities..

WS-Trust & WS-SecureConversation

WS-Trust

Enables applications to construct trusted SOAP message exchanges

- Trust represented through the exchange and brokering of security tokens
- The specification provides a protocol by which: issue, renew and validate security tokens

WS-SecureConversation

Enables secure *conversations* between two or more Web Services

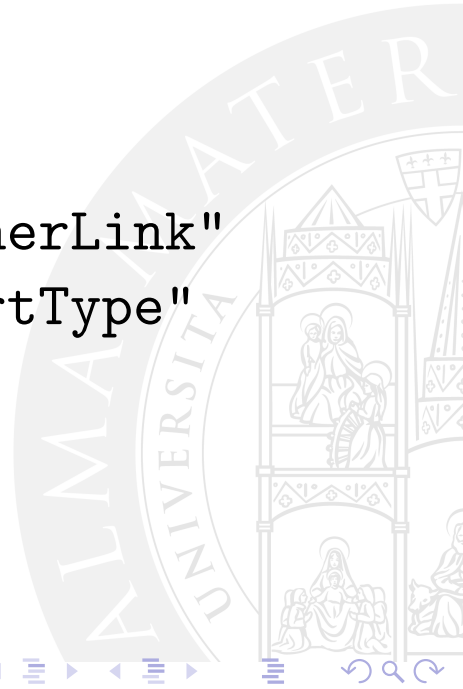
- Built on top of WS-Security and WS-Trust
- Use of security contexts and derived keys to enable a secure conversation

Web Service Business Process Language (WS-BPEL)

- Orchestration language that provides a means to formally specify business processes and interaction protocols
  - Extends the Web Services interaction model
  - Composition is based on pre-modelled workflow
- Basic activities
  - Invoke, receive, assign
- Structured activities
  - Sequence, flow, foreach

A BPEL Business Process Example I

```
<process name="MergedProductionWorkflow">
   <import />
   <partnerLinks />
   <variables />
   <sequence>
      <!-- Step 1: The process-execution layer
         receives a production order -->
      <receive name="receiveProductionOrder"
         partnerLink="productionManagementPartnerLink"
         portType="prod:ProductionManagementPortType"
         operation="receiveProductionOrder"
         createInstance="yes"
         variable="productionOrder"/>
```



A BPEL Business Process Example II

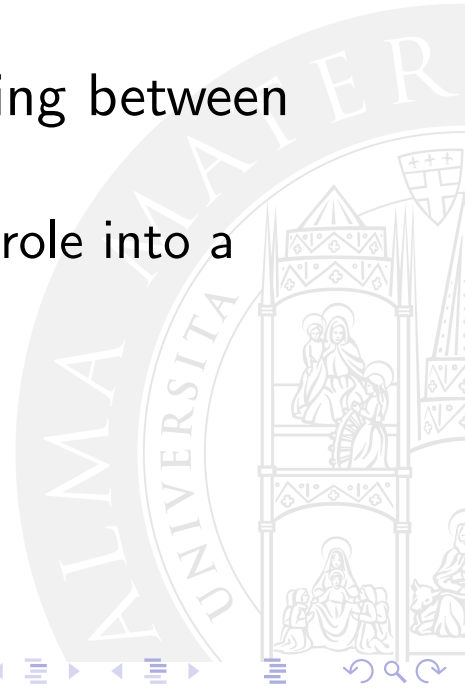
```
<invoke name="getListOfProductionItems"
    partnerLink="productionItemsListPartnerLink"
    portType="list:ProductionItemsListPortType"
    operation="getListOfProductionItems"
    inputVariable="productionOrder"
    outputVariable="productionItems">
</invoke>
```

A BPEL Business Process Example III

```
        ...
        <!-- Response back to requesting client application-->
        <reply name="replyPurchaseOrder"
           partnerLink="productionManagementPartnerLink"
           portType="prod:ProductionManagementPortType"
           operation="receiveProductionOrder"
           variable="responseMessage" />
        </sequence>
</process>
```

WS-BPEL: Remarks

- Web Service composition and orchestration realised by means of an offline business plan
- Does this behaviuor respect the service-orientation principles?
  - Not completely...
- Such an approach (*implicitly*) promotes control coupling between services
  - i.e. Web Service developed already w.r.t. a particular role into a orchestration scenario: no reuse



Semantic Web in a Nutshell

Tim Berners-Lee vision [Berners-Lee and Fischetti, 1999]

I have a dream for the Web in which computers become capable of analyzing all the data on the Web - the content, links, and transactions between people and computers. A *Semantic Web*, which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The *intelligent agents* people have touted for ages will finally materialize [Berners-Lee and Fischetti, 1999]

- Goal: *use* and *reason upon* all the available data on the internet automatically
- By extending the current web with knowledge - semantic information - about the content (i.e. data about the data, meta-data)

Semantic Web Services I

Introduction

- Researches area, in the ambit of the Semantic Web, that aims to introduce semantics aspects into the world of Web Service
- Objective: enable WSs to communicate via *machine-readable* data
- Match regarding *concepts*, *not simply signatures*
  - Composition/discovery driven by the *meaning* of the required data/functionalities

Foundations

- Ontologies: rigorous and formal description of a domain (e.g. OWL)
- Definition of the WS behavior (e.g. OWL-S, WSMO)
  - By means of Input, Output, Preconditions, Effects (IOPE)
- *Software agents* able to compose suitable WSs w.r.t the user goal

Outline

RESTFul Web Services: Why? I

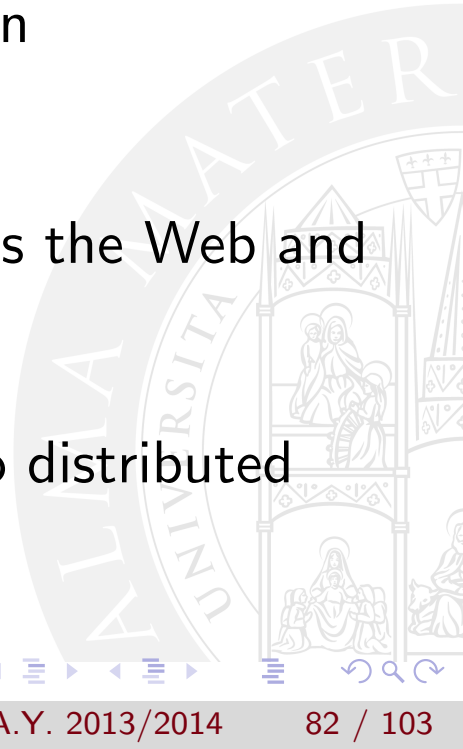
- In ten years the Web has changed the way we live, but it's got more change left to give
- Rooted on three main technologies

  HTTP as the transport protocol

  XML (HTML/XHTML) for data representation

  URIs for referring to *resources*

- The above technologies are powerful enough to give us the Web and the applications we use on it
- It is almost time to seriously start applying its rules to distributed programming



RESTFul Web Services: Why? II

Web's potential for distributed programming has been overlooked

The Web is a simple, ubiquitous, yet *overlooked* platform for distributed programming [Richardson and Ruby, 2007]

- Most of today's Web Services have nothing to do with the Web
  - In opposition to its simplicity, they espouse a heavyweight architecture for realising distributed applications
- It has to be that way?
- It is time to put the *Web* back into *Web Services*

REST

The original definition

*R*epresentational *S*tate *T*ransfer (REST) style is an abstraction of the architectural elements within a distributed hypermedia system [Fielding, 2000]

- Data and functionalities are considered *resources*
  - Accessed using URIs
- The resources are acted upon well-defined operations
  - HTTP methods: GET, POST, PUT, DELETE
- Client/server architecture designed to use a *stateless* communication protocol (HTTP)
- Clients/servers exchange representations of *resources* by using a standardized interface and protocol

Resource-Oriented Architecture in a Nutshell[Richardson and Ruby, 2007]

Four concepts:

- Resources
- Their names (URIs)
- Their representations
- The links between them

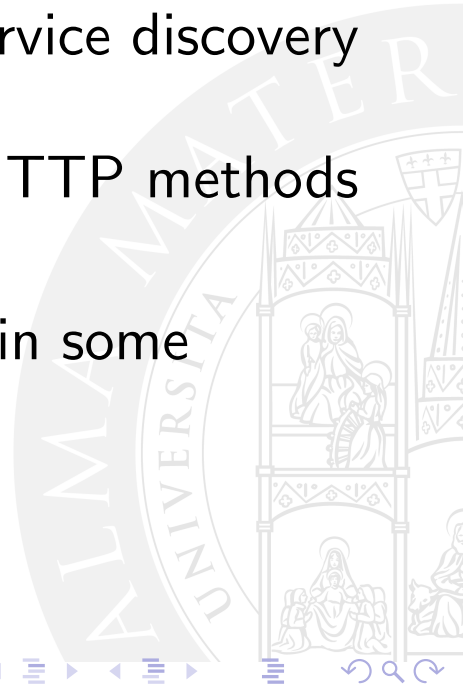
Four properties:

- Addressability
  - via URIs
- Statelessness
- Connectedness
  - resources connection through hyper-links
- A uniform interface
  - resource management through HTTP methods

RESTful Web Service I

- RESTful WSs are based on Resource-Oriented Architecture (ROA)
  - See [Richardson and Ruby, 2007] for details
- A RESTful Web Service exposes a set of resources identifying the targets of the interaction with its clients
- URIs provide an addressing space for resources and service discovery
- Uniform interface: Resources manipulation via fixed HTTP methods

PUT	creates a new resource
GET	retrieves the current state of a resource in some representation
DELETE	deletes an existing resource
POST	transfers a new state onto a resource

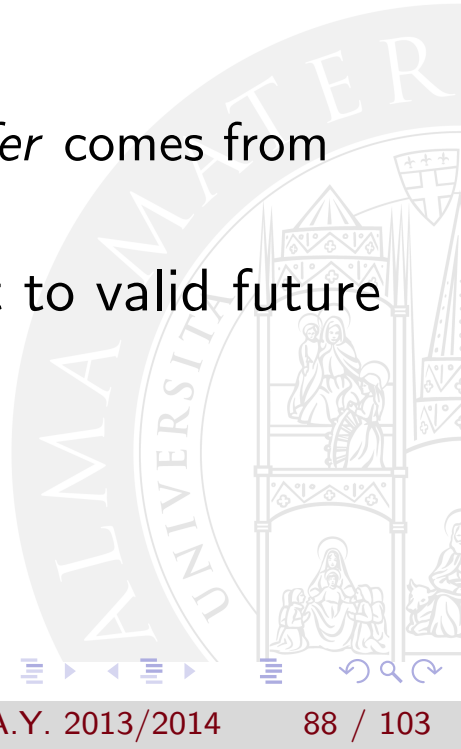


RESTful Web Service II

- Self-descriptive messages
  - Resources are decoupled from their representation
  - Content accessible in a variety of formats
    - HTML, XML, plain text, PDF, JPEG, JSON, ...
- Meta-data about the resource is available and used for
  - Caching control
  - Transmission errors detection
  - Appropriate representation format negotiation
  - Authentication or access control
- Every interaction with a resource is stateless
  - Like in the SOA case messages are self-contained

RESTful Web Service III

- Stateful interactions on the concept of *explicit state transfer*
  - Clients manipulate resource state by sending a representation as part of a PUT or POST request
  - Server manipulates client state sending representations in response to the client's GET requests
  - This is where the name *Representational State Transfer* comes from
- State can be embedded in response messages to point to valid future states of the interaction



RESTful Web Service Tools for Java

- JAX-RS specification (recommended)
  - Standard Java programming model (= set of API) for RESTful Web Services
  - Several implementations exist
    - See [Little, 2008] for a comparison
    - Jersey is the GlassFish implementation [Jersey, 2011]
- JAX-WS
  - Exploiting WSDL 2.0 for defining the REST Web Services
  - Usable, but not so used

JAX-RS in a Nutshell

- Really similar (in the spirit) to its *big brother* JAX-WS
- Provides an annotation-based model to simplify the development of a restful Web Service
  - ROA perspective
- Plain Old Java Object (POJO) can be easily exposed as a Web service

JAX-RS in Action

```
@Path("helloworld")
public class HelloWorld {
    @Context
    private UriInfo context;
    /** Creates a new instance of HelloWorld */
    public HelloWorld() {
    }

    /**
     * Retrieves representation of an instance of helloWorld.HelloWorld
     * @return an instance of java.lang.String
     */
    @GET
    @Produces("text/html")
    public String getHtml() {
        return "<html><body><h1>"+ msg +"Hello, World!!</h1></body></html>";
    }

    /**
     * PUT method for updating or creating an instance of HelloWorld
     * @param content representation for the resource
     * @return an HTTP response with content of the updated or created resource.
     */
    @PUT
    @Consumes("text/html")
    public void putHtml(String content) {
    }
}
```

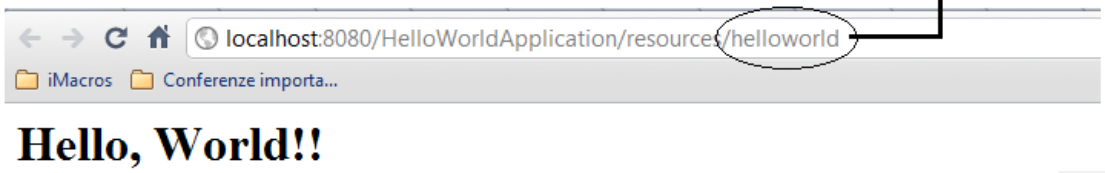


Figure : A REST Web Service printing in output the classical "Hello world!".

Other RESTful Web Service Tools

- Ruby on Rails
  - http://rubyonrails.org/
- .NET based tools: Microsoft WCF
  - http://msdn.microsoft.com/en-us/netframework/aa663324
- Python based tools
  - http://www.djangoproject.com/
  - http://cherrypy.org/
- ...

Outline

Summing Up

- Web Services are one of the reference technology for building distributed systems
- Two different architectural styles exist
  - SOA vs ROA [Pautasso et al., 2008]
- An ongoing *"holy war"* between the two styles
  - With strong supporters/experts in both sides
  - Often driven by not so strong/valid arguments
  - Difficult to provide a rigorous evaluation
- The question is: *which architecture should I use?*

SOA vs. ROA (Andrea Santi's Opinion) I

SOA Benefits

- SOA is weighted by standards designed to promote interoperability
  - WSDL for describing the WS functionalities/interfaces
  - WS-* for high level functionalities support
- Therefore better suited for
  - Enterprise and B2B solutions
  - Composition and integration of WSs & existing applications

SOA vs. ROA (Andrea Santi's Opinion) II

ROA benefits

- The main advantage of ROA is ease of implementation, agility of the design, and the lightweight approach to things
- REST is a lightweight solution as simple as the Web
  - No standards at all (except HTTP, XML, URI)
- Lower entry barrier
- Web Services accessible also from **mobile devices**
  - That is not the case for SOA WSs
- Simplicity is its siren call
  - Being heard even in the far corners of corporate data centers

SOA vs. ROA (Andrea Santi's Opinion) III

In the overall

- There is not a real winner yet
- A lot of developers and WS have turned to the ROA side
  - Because it seems faster, cheaper and easier
- But standard-less development can require more investment
  - To maintain and manage
  - In learning data formats (are you using XML? JSON? CSV?)
  - In learning service descriptions

SOA vs. ROA (Andrea Santi's Opinion) IV

- Use ROA when
  - You needs something up-and-running quickly...
  - ...with good performance and low overhead
  - Web Services easily exploitable by mobile devices & simple clients
    - e.g. AJAX/Javascript-based
- Use SOA when you need a distributed application with
  - Explicit definition of Web Services contacts
  - Support for high level functionalities (WS-*)

References I

Berners-Lee, T. and Fischetti, M. (1999).
Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor.
Harper San Francisco.

Erl, T. (2005).
Service-Oriented Architecture: Concepts, Technology, and Design.
Prentice Hall PTR, Upper Saddle River, NJ, USA.

Fielding, R. T. (2000).
Architectural Styles and the Design of Network-based Software Architectures.
PhD thesis, University of California, Irvine, CA, USA.

References II


Jersey (2011).
Jersey home page.
http://jersey.java.net/.


Little, M. (2008).
A comparison of JAX-RS implementations.
http://www.infoq.com/news/2008/10/jaxrs-comparison.


Oracle, S. M. . (2011).
The Java EE 6 tutorial, part 3: Web Services.
http://download.oracle.com/javaee/6/tutorial/doc/bnayk.html.

References III


Pautasso, C., Zimmermann, O., and Leymann, F. (2008).
RESTful Web Services vs. "big" Web Services: Making the right architectural decision.
In *17th International Conference on World Wide Web (WWW '08)*, pages 805–814, New York, NY, USA. ACM.


Richardson, L. and Ruby, S. (2007).
RESTful Web Services.
O'Reilly.


Sun/Oracle (2004).
Java Metro reference site.
`http://metro.java.net/`.

References IV


The Apache Software Foundation (2004).
Axis 2 reference site.
http://ws.apache.org/axis2.

Web Services

Distributed Systems

Sistemi Distribuiti

Andrea Omicini *after Andrea Santi*
andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2013/2014